Relazione progetto di Programmazione ad Oggetti : *ItemsCollection*

Alessandro Mazzariol 2101050

Indice

1 Introduzione

ItemsCollection è un programma che gestisce una collezione di oggetti di vario tipo, il suo punto di forza è la griglia di copertine degli oggetti nella home page. Da questa visualizzazione generale, è possibile accedere ad una visione dettagliata dell'oggetto selezionato, in quest'ultima è poi possibile effettuare le operazioni di servizio come l'eliminazione e la modifica l'oggetto. L'applicativo permette inoltre di creare un nuovo oggetto nella sezione apposita, accessibile tramite la toolbar. La barra di ricerca permette di cercare un titolo con una ricerca realtime, quest'ultima è collegata ai bottoni posti in alto che permettono invece di filtrare la griglia per i possibili tipi di oggetti.

2 Modello logico

2.1 Le classi Items

Il modello logico si basa su due parti interconnesse tra loro (non visibile tramite diagramma per questioni di visualizzazione), la gestione dei tipi di item e la gestione dei form di creazione degli Item. La prima comprende le classi che descrivono gli Items, come riportato in Figura 1, partendo da una classe astratta *AbstractItem* che rappresenta le informazioni in comune tra tutti gli item ricercabili, ovvero un Titolo, Immagine di Copertina, Descrizione e Anno, con i relativi metodi getter e setter. Vi sono poi le singole classi concrete, in questo applicativo ci sono tre classi concrete, *Book, Game* e *Music*; è presente inoltre una quarta classe concreta *Collezione* che funge da contenitore dei singoli Item e sfrutta il *Singleton Pattern*. Per permettere un'efficiente gestione di memoria ho scelto di usare un tipo di puntatore offerto da Qt *QSharedPointer* che consentiva inoltre una maggiore flessibilità rispetto ai comuni puntatori. Per risolvere la questione del Polimorfismo non banale ho optato per il *design pattern Visitor*, a questo scopo sono state realizzate le classi *AbstractVisitor e AbstractConstVisitor* con l'unica differenza che la seconda non modifica l'oggetto che è stato visitato, per funzionare correttamente sono presenti di conseguenza i metodi accept nelle relative classi di Item.

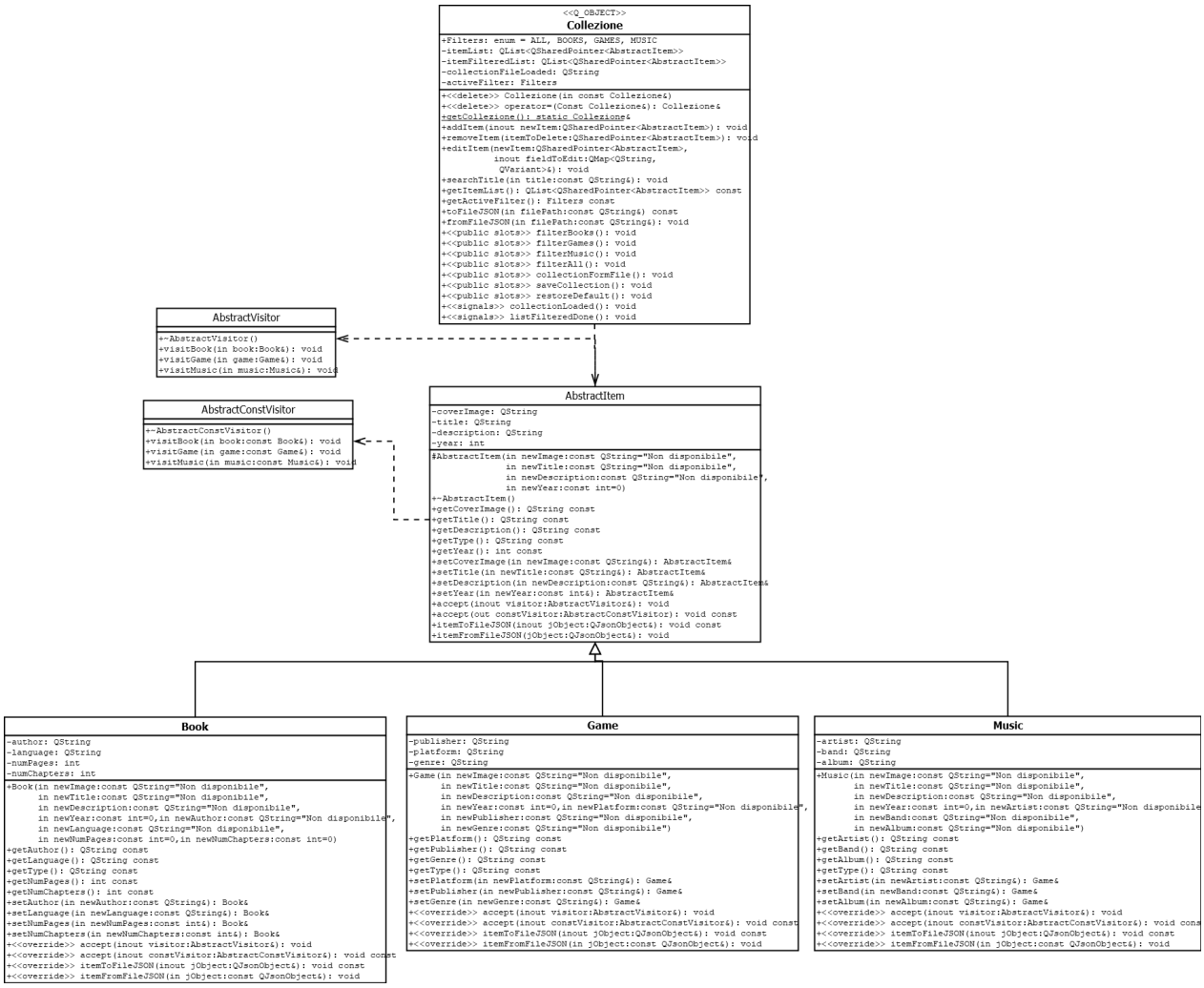

Figure 1: Diagramma delle classi Items del modello

2.2 Le classi FormCreation

Come introdotto in precedenza, la seconda parte riguarda la gestione dei Form di creazione e modifica, che meritano una propria spiegazione, il modello, come visto in Figura 2, parte da una classe astratta *ItemCreationForm*, questa classe è il fulcro per quanto riguarda la creazione degli item, comprendendo gli attributi comuni a tutti i *FormCreation*, tramite l'utilizzo del metodo *CreateDerivedItem* che crea un oggetto del tipo scelto dall'utente, quando quest'ultimo seleziona uno dei tre pulsanti disponibili nella pagina di creazione. Anche qui è stata usata la tipologia di puntatori di Qt *QSharedPointer*, per il polimorfismo sono stati creati cinque visitor differenti *FormVisitor, FormConstVisitor, VisitorReadForm, VisitorReadItem, VisitorCreateForm*, dove i primi due sono riconducibili ai visitor della prima parte del modello e rimanenti differiscono per il fatto che *VisitorReadForm e VisitorReadItem* lasciano invariato l'oggetto visitato garantendo la *const correctness*. Ogni classe *FormCreation* è definita con la macro *Q_OBJECT* per permettere l'utilizzo di signals e slot.

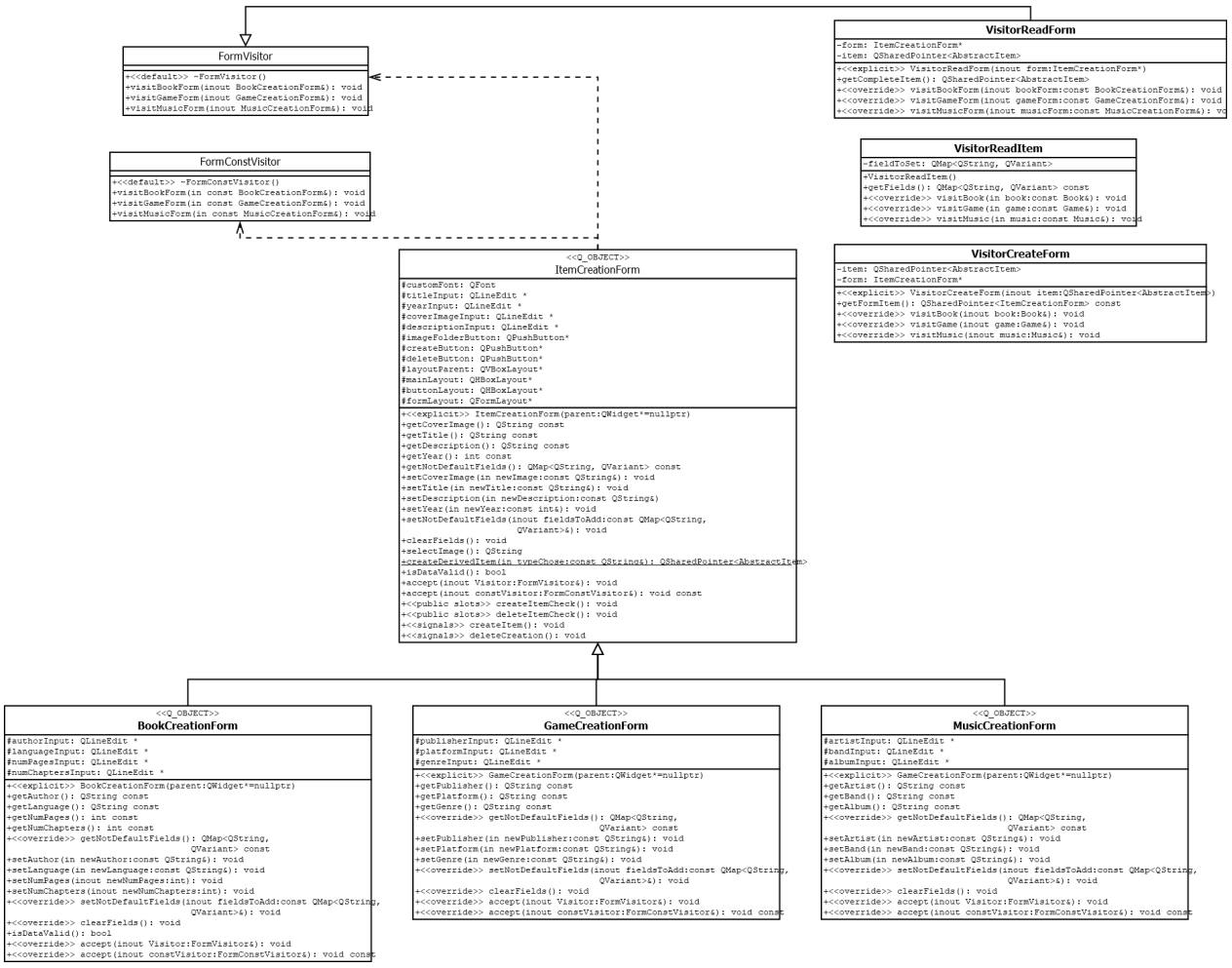

Figure 2: Diagramma delle classi FormCreation del modello

3 Polimorfismo

Come detto in precedenza il polimorfismo è presente usando il *design pattern Visitor* il quale consente di separare la logica delle operazione eseguibili sugli oggetti Item dalla loro struttura interna, questo approccio è stato fondamentale per gestire la relazione tipologia di item e form associato ad esso, per questo motivo sono stati implementati tre visitor ognuno con uno scopo ben definito:

- *VisitorReadForm*: legge i dati inseriti nei form di creazione *BookCreationForm, GameCreationForm, MusicCreationForm* e genera un oggetto concreto *Book, Game, Music* con i dati all'interno che viene successivamente salvato nel JSON tramite i metodi virtuali presenti nelle classi concrete di *AbstractItem, itemToFileJSON*.

- *VisitorCreateForm*: agisce in direzione opposta al visitor precedente, a partire da un *AbstractItem*, determina il tipo concreto e crea il relativo form di creazione, questo quindi permette di evitare dynamic_cast o controlli espliciti sul tipo.

- *VisitorReadItem*: questo visitor accede invece ai campi specifici di un *AbstractItem* e raccoglie i dati specifici deducendo il tipo concreto, inserendole in una *QMap<QString, QVariant>*. Questo visitor viene usato quando l'utente seleziona un elemento nella griglia della home page decidendo di visualizzare l'item nel dettaglio.

Questo pattern quindi mi permette di simulare la tecnica del *double dispatch* eseguendo comportamenti in base al tipo dell'oggetto ricevente e in base al tipo di visitatore. In aggiunta vi sono controlli tramite il dynamic_cast di Qt per i shared_pointer *qSharedPointerDynamicCast* usati principalmente per le parti di GUI, avendo soltanto tre tipologia di classi concrete non era necessario creare un visitor apposito, questi controlli sono presenti nelle classi *ItemPageEditUI, ItemPageUI e ItemWidgetUI* che non sono presenti nel diagramma UML.

4 Persistenza dei Dati

Per la persistenza dei dati viene utilizzato il formato JSON, gli oggetti salvati all'interno sono semplici associazioni chiave-valore, la serializzazione viene gestita con l'aggiunta del campo "Tipo", assieme al codice del programma è presente un esempio di struttura JSON, in particolare è presente "tems.json" nella cartella relativa "JSON". Insieme a questo file sono stati messi degli asset per visualizzare i relativi oggetti nella cartella al percorso "assets/cover".
Qualora un Item fosse stato creato privo di immagine di copertina, viene inserita un immagine di default presente nella cartella al percorso "assets/cover/notAvailable/ImageNotAvailable.jpg"

5 Funzioni Implementate

Le funzionalità implementate riguardano la gestione di tre tipologie di item, la conversione e salvataggio in un formato JSON, ricerca di Item in base al titolo e/o per tipologia. Per quanto riguarda la GUI, invece, ItemsCollection è dotato di una *QToolBar* con quattro possibili scelte:

- Home: porta l'utente nella home page dove viene visualizzata la collezione in una griglia al centro della pagina
- About: porta nella pagina about del progetto con qualche informazione a riguardo
- Collezione: chiama una finestra dell'esplora risorse, che consente l'utente di caricare un file collezione personale
- Crea: porta nella sezione di creazione di un oggetto, nella pagina di creazione vengono presentati tre pulsanti scegliendo uno di essi verrà poi mostrato il form da completare
- Exit: mostra una finestra di dialogo che porta al termine dell'esecuzione dell'applicativo

In tutto l'applicativo troviamo finestre di dialogo per confermare la scelta in questi momenti specifici:

- Nella toolbar quando l'utente clicca il pulsante EXIT
- Quando l'utente annulla la creazione di un oggetto durante la visualizzazione dell'*ItemCreationForm*
- Quando l'utente tenta di elimare l'item selezionato durante la visualizzazione della pagina completa dell'item scelto
- Quando l'utente vuole annullare la modifica di un item durante la visualizzazione della pagina completa in modalità EDIT

Infine all'interno dell'applicativo l'utente durante la creazione dell'item viene informato con una finestra di warning qualora i suoi dati inseriti non rispettassero queste condizioni:

- Titolo mancante
- Anno inserito è minore o uguale a 0 oppure sono stati inseriti caratteri alfabetici
- Durante la creazione di un libro, il numero di pagine e di capitoli sono minori o uguali a 0 oppure presentano caratteri alfabetici

Nella Home page le copertine presenti nella griglia quando vengono selezionate portano alla pagina con i dettagli completi, all'interno di questa visualizzazione vi sono i pulsanti per l'eliminazione dell'item e per la modifica che trasforma i *QLabel* in *QLineEdit* permettendo quindi l'inserimento di nuovi dati. Se l'utente clicca di "Si" questo comporta la rimozione non solo all'interno della collezione locale, ma anche nella struttura nel JSON.
I pulsanti vicino alla barra di ricerca, "Salva" e "Ripristina", permettono rispettavemente di salvare la collezione caricata in quel momento, in un file JSON chiamato Items.Json e di ripristinare la collezione di base, che è quella contenuta nel percorso file "JSON/default". L'applicativo ha una propria icona dell'eseguibile presente nella cartella "assets/icon". La scelta di usare il singleton pattern per la classe *Collezione* permette di avere una visualizzazione runtime e quindi più fluida degli item durante la ricerca e il filtraggio, eliminando quindi la necessità di avere un pulsante per la conferma.

6 Rendicontazione delle ore

Attività	**Ore Previste**	**Ore Effettive**
Studio e Progettazione	10	15
Sviluppo del codice del modello	10	15
Studio del framework Qt	10	13
Sviluppo del codice della GUI	10	8
Test e Debug	5	6
Stesura della relazione	5	3
Totale	50	60

Il monte ore è stato superato per via della difficoltà iniziale nel progettare e dello sviluppo del modello, ho potuto però risparmiare un po' di tempo nello sviluppo della GUI avendo in passato già avuto a che fare con creazione di interfaccia grafica di questa tipologia. Lo studio del framework Qt ha richiesto più tempo per capire come rendere il codice più flessibile e adatto alle mie esigenze e soprattuto per rendere il codice più efficiente in questione di velocità.